ELSEVIER

Theoretical Computer Science 275 (2002) 283–310

Theoretical Computer Science

www.elsevier.com/locate/tcs

Program schemes, arrays, Lindström quantifiers and zero-one laws ☆

Iain A. Stewart *, [1]

Department of Mathematics and Computer Science, University of Leicester, Leicester LE1 7RH, UK

Received November 1999; revised January 2001; accepted March 2001
Communicated by W. Thomas

Abstract

We characterize the class of problems accepted by a class of program schemes with arrays, NPSA, as the class of problems defined by the sentences of a logic formed by extending first-order logic with a particular uniform (or vectorized) sequence of Lindström quantifiers. A simple extension of a known result thus enables us to prove that our logic, and consequently our class of program schemes, has a zero-one law. However, we use another existing result to show that there are problems definable in a basic fragment of our logic, and so also accepted by basic program schemes, which are not definable in bounded-variable infinitary logic. As a consequence, the class of problems NPSA is not contained in the class of problems defined by the sentences of partial fixed-point logic even though in the presence of a built-in successor relation, both NPSA and partial fixed-point logic capture the complexity class **PSPACE**.

Keywords: Finite model theory; Complexity theory; Program schemes; Lindström quatifiers; Zero-one laws

1. Introduction

This paper is a continuation of the study of the classes of problems captured by different classes of program schemes (in this study, the particular emphasis is on a comparison with the classes of problems defined by the sentences of well-known logics from finite model theory). *Program schemes* form a model of computation that is

☆ An extended abstract of this paper appeared in Proc. Computer Science Logic, Lecture Notes in Computer Science, Vol. 1683, Springer-Verlag (1999) 374–388.

* Tel.: +44-116-2525356; fax: +44-116-2523915.
E-mail address: ias4@leicester.ac.uk (I.A. Stewart).

[1] Supported by EPSRC Grants GR/K 96564 and GR/M 12933.

PII: S0304-3975(01)00183-9

amenable to logical analysis yet is closer to the general notion of a program than a logical formula is. Program schemes were extensively studied in the 1970s (for example, see [3, 7, 16, 35]), without much regard being paid to an analysis of resources, before a closer complexity analysis was undertaken in, mainly, the 1980s (for example, see [24, 26, 44]). There are connections between program schemes and logics of programs, especially dynamic logic [9, 30]. One might also view many query languages from database theory as classes of program schemes, although query languages tend to operate on relations as opposed to individual elements (for example, see the *while* language from [1, 4, 5] and the language BQL from [4, 33]).

One of the most basic classes of program schemes is that obtained by allowing assignments, while instructions with quantifier-free tests and non-determinism. In [6], the relative expressibilities of this class of program schemes, NPS(1), in the presence of a built-in successor, a built-in linear-order, a built-in multiplication, a built-in addition and combinations of such were completely classified. It was shown in [2] that NPS, an extension of NPS(1) obtained by allowing universally quantified program schemes to appear as tests in while instructions, is none other than transitive closure logic and that a computational analysis (as opposed to a model-theoretic, and in particular a game-theoretic, analysis) of such program schemes yields proper infinite hierarchies within NPS (and so within transitive closure logic). Also in [2], the class of program schemes obtained from NPS by allowing additional access to a stack, NPSS, was shown to be none other than path system logic (which had previously been shown to be none other than stratified fixed point logic and stratified Datalog [22, 28]), and again a computational analysis of the program schemes of NPSS was shown to yield proper infinite hierarchies within NPSS (and so within path system logic). Subsequently, in [43], a detailed analysis of certain program schemes of NPSS yielded that any polynomial-time problem involving strongly connected locally ordered digraphs, connected planar embeddings or triangulations (that is, planar graphs embeddable in the plane so that every face is a cycle of length 3) can be defined in (a proper fragment of) path system logic (without any built-in relations).

The results mentioned above show that the study of program schemes is intimately related with more mainstream logics from finite model theory. In [38], program schemes allowing assignments, while instructions with quantifier-free tests, non-determinism and access to arrays were studied but only in the presence of a built-in successor relation (the class of problems accepted by such program schemes was shown to be **PSPACE**). It is with these program schemes and their extensions, obtained by allowing universally quantified program schemes to appear as tests in while instructions, that we are concerned in this paper but in the absence of any built-in relations; that is, the class of program schemes NPSA. Our class of program schemes NPSA is quite natural. It consists of the union of an infinite hierarchy of classes of program schemes

$$\mathrm{NPSA}(1) \subseteq \mathrm{NPSA}(2) \subseteq \mathrm{NPSA}(3) \subseteq \cdots.$$

The program schemes of NPSA(1) are built by allowing assignments, while instructions with quantifier-free tests, non-determinism and access to arrays (full details follow later). The program schemes of NPSA(2) are built from program schemes of NPSA(1) by universally quantifying free variables. The program schemes of NPSA(3) are built as are the program schemes of NPSA(1) except that tests in while instructions can be program schemes of NPSA(2). The program schemes of NPSA(4) are built from program schemes of NPSA(3) by universally quantifying free variables; and so on.

What is crucial is our definition of the semantics. Consider, for example, a while instruction in a program scheme ρ of NPSA(3) where the test is a program scheme ρ' of NPSA(2). In order to evaluate whether the test is true or not, the arrays from ρ are not 'passed over' to the program scheme ρ': the evaluation of ρ' has no access to the arrays of ρ. After evaluation of ρ' has been completed, the computation of the program scheme ρ resumes accordingly with its arrays having exactly the same values as they had immediately prior to the evaluation of ρ'. It is essentially our semantic definition that enables us to characterize the class of problems accepted by the program schemes of NPSA as the class of problems defined by the sentences of a logic $(\pm\Omega)^*$[FO] formed by extending first-order logic with a particular uniform (or vectorized) sequence of Lindström quantifiers (where this uniform sequence of Lindström quantifiers corresponds to a **PSPACE**-complete problem Ω). Moreover, we show that the logic $(\pm\Omega)^*$[FO] has a zero-one law; but not because it is a fragment of bounded-variable infinitary logic, as is so often the case in finite model theory, for we show that there are problems definable in NPSA (in NPSA(1) even) which are not definable in bounded-variable infinitary logic. Consequently, whilst both NPSA and partial fixed-point logic capture the complexity class **PSPACE** in the presence of a built-in successor relation, there are problems in NPSA which are not definable in partial-fixed point logic. If our semantics were such as to allow for universal quantification over arrays then we could simply guess a successor relation and hold our guesses in an array, use universal quantification to verify that the guessed relation was indeed a successor relation and subsequently use this guessed relation as our successor relation throughout. Consequently, we would have captured **PSPACE** and not the interesting logics (with zero-one laws but which are not fragments of bounded-variable infinitary logic) encountered in this paper.

Section 2 includes our preliminary definitions (with [13] serving as our basic reference text for finite model theory). In Section 3, we establish complete problems for NPSA(1) via quantifier-free first-order translations with two constants, and in Section 4, we use these completeness results to obtain our logical characterizations of NPSA. We then use a result due to Stewart to show that NPSA(1) (and so NPSA) is not contained in bounded-variable infinitary logic. In Section 5, we extend a result due to Dawar and Grädel and hence show that our logics from the previous section, and consequently NPSA, have a zero-one law. Finally, we present our conclusions and directions for further research.

2. Preliminaries

2.1. Logic

Ordinarily, a *signature* σ is a tuple $\langle R_1, \ldots, R_r, C_1, \ldots, C_c \rangle$, where each R_i is a relation symbol, of arity a_i, and each C_j is a constant symbol. However, we sometimes consider signatures in which there are no constant symbols; that is, *relational signatures*. *First-order logic over the signature* σ, $\mathrm{FO}(\sigma)$, consists of those formulae built from atomic formulae over σ using $\wedge$, $\vee$, $\neg$, $\forall$ and $\exists$; and $\mathrm{FO} = \bigcup\{\mathrm{FO}(\sigma)$: σ is some signature$\}$.

A *finite structure* $\mathscr{A}$ *over the signature* σ, or σ*-structure*, consists of a finite *universe* or *domain* $|\mathscr{A}|$ together with a relation R_i of arity a_i, for every relation symbol R_i of σ, and a constant $C_j \in |\mathscr{A}|$, for every constant symbol C_j (by an abuse of notation, we do not distinguish between constants or relations and constant or relation symbols). A finite structure $\mathscr{A}$ whose domain consists of n distinct elements has *size* n, and we denote the size of $\mathscr{A}$ by $|\mathscr{A}|$ also (this does not cause confusion). We only ever consider finite structures of size at least 2, and the set of all finite structures of size at least 2 over the signature σ is denoted $\mathrm{STRUCT}(\sigma)$. A *problem* over some signature σ consists of a subset of $\mathrm{STRUCT}(\sigma)$ that is closed under isomorphism; that is, if $\mathscr{A}$ is in the problem then so is every isomorphic copy of $\mathscr{A}$. Throughout, all our structures are finite.

2.2. Lindström quantifiers

We are now in a position to consider the class of problems defined by the sentences of FO: we denote this class of problems by FO also, and do likewise for other logics. It is widely acknowledged that, as a means for defining problems, first-order logic leaves a lot to be desired especially when we have in mind developing a relationship between computational complexity and logical definability. In particular, every first-order definable problem can be accepted by a logspace deterministic Turing machine yet there are problems in the complexity class **L** (logspace) which cannot be defined in first-order logic (one such being the problem consisting of all those structures, over any signature, that have even size). Consequently, we now illustrate one way of increasing the expressibility of FO: we augment FO with a uniform or vectorized sequence of Lindström quantifiers, or operator for short (the reader is referred to [13] for a fuller exposition on the limitations of FO and on a number of different methods, including this one, for increasing the expressibility of FO).

Our illustration uses an operator derived from a problem whose underlying instances can be regarded as path systems. A *path system* consists of a finite set of *vertices* and a finite set of *rules*, each of the form (x, y, z), where x, y and z are (not necessarily distinct) vertices. There is a unique distinguished vertex called the *source* and a unique distinguished vertex called the *sink*. The set of *accessible vertices* in any path system is built as follows. Initially, the source is deemed to be accessible and new vertices are shown to be accessible by *applying* the rules via: if x and y are accessible (with

possibly $x = y$) and there is a rule (x, y, z) then z becomes accessible. The *path system problem* consists of all those path systems for which the sink is accessible from the source, and it was the first problem to be shown to be complete for the complexity class **P** (polynomial-time) via logspace reductions [8].

We encode the path system problem as a problem over the signature σ_{3++} which consists of the relation symbol R of arity 3 and the constant symbols *source* and *sink*. A σ_{3++}-structure $\mathscr{P}$ can be thought of as a path system where the vertices of the path system are given by $|\mathscr{P}|$, the source is given by *source*, the sink is given by *sink* and the rules of the path system are given by $\{(x, y, z)$: $R(x, y, z)$ holds in $\mathscr{P}\}$. Hence, we define the problem PS as

$$\{\mathscr{P} \in \mathrm{STRUCT}(\sigma_{3++})\text{: the vertex } \textit{sink} \text{ is accessible from the vertex } \textit{source} \text{ in the path system } \mathscr{P}\}.$$

Let us return to increasing the expressibility of FO. Corresponding to the problem PS is an operator of the same name. The logic $(\pm\mathrm{PS})^*[\mathrm{FO}]$, or *path system logic*, is the closure of FO under the usual first-order connectives and quantifiers and also the operator PS, with PS applied as follows.

Given a formula $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in (\pm\mathrm{PS})^*[\mathrm{FO}]$ over some signature σ, where the variables of the k-tuples $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, for some $k \geqslant 1$, are all distinct and free in φ, the formula Φ defined as $\mathrm{PS}[\lambda\mathbf{x}, \mathbf{y}, \mathbf{z}\varphi](\mathbf{u}, \mathbf{v})$, where $\mathbf{u}$ and $\mathbf{v}$ are k-tuples of (not necessarily distinct) constant symbols and variables, is also a formula of $(\pm\mathrm{PS})^*[\mathrm{FO}]$. The free variables of Φ are those variables in $\mathbf{u}$ and $\mathbf{v}$ together with the free variables of φ different from those in the tuples $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. If Φ is a sentence then it is interpreted in a structure $\mathscr{A} \in \mathrm{STRUCT}(\sigma)$ as follows. We build a path system with vertex set $|\mathscr{A}|^k$ and set of rules

$$\{(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in |\mathscr{A}|^k \times |\mathscr{A}|^k \times |\mathscr{A}|^k\text{: } \varphi(\mathbf{a}, \mathbf{b}, \mathbf{c}) \text{ holds in } \mathscr{A}\},$$

and say that $\mathscr{A} \models \Phi$ if, and only if, the sink $\mathbf{v}$ is accessible in this path system from the source $\mathbf{u}$ (the semantics can easily be extended to arbitrary formulae of $(\pm\mathrm{PS})^*[\mathrm{FO}]$: see, for example, [13] for a more detailed semantic definition of operators such as PS). Note that there is nothing special about the problem PS: any problem can be converted into an operator and used to extend first-order logic. Syntactically, such logics are very similar although their semantics depend on the operator in hand.

It is indeed the case that we have increased expressibility as we can define problems in $(\pm\mathrm{PS})^*[\mathrm{FO}]$ which cannot be defined in FO (a simple Ehrenfeucht–Fraïssé game shows that PS is not definable in FO: see [13] for more on such games). In the presence of a built-in successor relation, we can obtain a precise complexity-theoretic characterisation of the problems definable in $(\pm\mathrm{PS})^*[\mathrm{FO}]$. We say that we have a *built-in successor relation* if no matter over which signature we happen to be working, there is always a binary relation symbol *succ* and two constant symbols 0 and *max* available such that this relation symbol *succ* is always interpreted as a successor relation, of the form $\{(a_0, a_1), (a_1, a_2), \ldots, (a_{n-2}, a_{n-1})\}$, in a structure of size n, where all the

a_i's are distinct and $a_0 = 0$ and $a_{n-1} = max$. Note that whether a structure satisfies a sentence, in which the relation symbol *succ* or the constant symbols 0 or *max* appear, might depend upon the particular successor relation chosen as the interpretation for *succ*. Consequently, we only consider those sentences of $(\pm\mathrm{PS})^*[\mathrm{FO}]$ with a built-in successor relation that define problems as being well formed; that is, those sentences for which satisfaction is independent of the particular interpretation chosen for *succ*. We denote the logic $(\pm\mathrm{PS})^*[\mathrm{FO}]$ with a built-in successor relation by $(\pm\mathrm{PS})^*[\mathrm{FO}_s]$ (and adopt a similar notation for other logics). As to whether $(\pm\mathrm{PS})^*[\mathrm{FO}_s]$ should really be called a logic is highly debatable (for example, it is undecidable as to whether a sentence of $(\pm\mathrm{PS})^*[\mathrm{FO}_s]$ is *order-invariant*, i.e., satisfies the property we want as regards *succ*, and so this 'logic' does not have a recursive syntax) and the reader is referred to [13, 34] for a detailed discussion of this and related points. However, it turns out that a problem is in the complexity class **P** if, and only if, it can be defined by a sentence of $(\pm\mathrm{PS})^*[\mathrm{FO}_s]$ [40].

Our notation for $(\pm\mathrm{PS})^*[\mathrm{FO}]$ is such that $\pm$ denotes the fact that applications of the operator PS can appear within the scope of negation signs and * denotes the fact that we are allowed to nest applications of PS as many times as we like. The fragment $(\pm\mathrm{PS})^k[\mathrm{FO}]$, for some $k \geqslant 1$, is obtained by allowing at most k nestings of applications of PS, and the fragment $\mathrm{PS}^k[\mathrm{FO}]$ is obtained by further disallowing any application of PS to appear within the scope of a negation sign.

In [40], it was shown that there is a very restricted normal form for sentences of $\mathrm{PS}^1[\mathrm{FO}_s]$. This normal form is such that any problem in $\mathbf{P} = (\pm\mathrm{PS})^*[\mathrm{FO}_s]$ can be defined by a sentence of $\mathrm{PS}^1[\mathrm{FO}_s]$ of the form

$$\mathrm{PS}[\lambda \mathbf{x}, \mathbf{y}, \mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})](\mathbf{0}, \mathbf{max}),$$

where: $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are k-tuples of distinct variables, for some $k \geqslant 1$; φ is a quantifier-free formula of FO_s; and $\mathbf{0}$ and $\mathbf{max}$ are k-tuples consisting of the constant symbols 0 and *max* repeated k times, respectively. (Note that in the absence of built-in relations, the hierarchy

$$\mathrm{PS}^1[\mathrm{FO}] \subset \mathrm{PS}^2[\mathrm{FO}] \subset \cdots$$

is proper [20].)

Saying that any problem in **P** can be described by a sentence of the above normal form is equivalent to saying that there is a *quantifier-free first-order translation with successor* from any problem in **P** to the problem PS; that is, PS is complete for **P** via quantifier-free first-order translations with successor. The reader is referred to [13] for more on logical translations where they go under the name of logical interpretations. However, in [13] logical translations involving only relational signatures are considered. If the target problem of a logical translation, PS above, is over a signature containing constant symbols then we assume that such constant symbols are specified by an appropriate tuple of other constant symbols (as is the case in the normal form result above where the constant symbols *source* and *sink* of σ_{3++} are specified by the

k-tuples **0** and **max**, respectively). Naturally, we have notions such as a *quantifier-free first-order translation* (where *succ* and 0 and *max* are not involved) and a *quantifier-free first-order translation with two constants* (where there are two built-in constants, which are always interpreted differently, but no built-in successor relation), and we can have logical translations involving formulae of other logics, not just quantifier-free first-order formulae.

A number of extensions of FO using operators corresponding to some problem Ω have been studied, as indeed has the whole notion of using such operators to extend FO. For example: numerous complexity classes have been 'captured' by (fragments of) logics of the form $(\pm\Omega)^*$[FO] (sometimes in which there are built-in relations) and a variety of problems have been shown to be complete for different complexity classes via different logical translations (see, for example, the papers [19, 25, 36, 37] and the references therein); proper infinite hierarchies have been established in logics of the form $(\pm\Omega)^*$[FO] (see, for example, [2, 20, 21]); logics of the form $(\pm\Omega)^*$[FO] have been shown to have zero–one laws [12] (we shall talk about zero–one laws for such logics in more detail later); and it has been shown that if there is a logic for **P** (where 'logic' is as in [13, 34]) then there is a logic for **P** of the form $(\pm\Omega)^*$[FO] [11].

2.3. Program schemes

An alternative and more computational means for defining classes of problems is to use program schemes. A *program scheme* $\rho \in \mathrm{NPSA}(1)$ involves a finite set $\{x_1, x_2, \ldots, x_k\}$ of *variables*, for some $k \geqslant 1$, and is over a signature σ. It consists of a finite sequence of *instructions* where each instruction, apart from the first and the last, is one of the following:

- an *assignment instruction* of the form '$x_i := y$', where $i \in \{1, 2, \ldots, k\}$ and where y is a variable from $\{x_1, x_2, \ldots, x_k\}$, a constant symbol of σ or one of the special constant symbols 0 and *max* which do not appear in any signature;
- an *assignment instruction* of the form '$x_i := A[y_1, y_2, \ldots, y_d]$' or '$A[y_1, y_2, \ldots, y_d] := y_0$', for some $i \in \{1, 2, \ldots, k\}$, where each y_j is a variable from $\{x_1, x_2, \ldots, x_k\}$, a constant symbol of σ or one of the special constant symbols 0 and *max* which do not appear in any signature, and where A is an array symbol of dimension d;
- a *guess instruction* of the form 'GUESS x_i', where $i \in \{1, 2, \ldots, k\}$; or
- a *while instruction* of the form 'WHILE φ DO $\alpha_1; \alpha_2; \ldots; \alpha_q$ OD', where φ is a quantifier-free formula of $\mathrm{FO}(\sigma \cup \{0, max\})$, whose free variables are from $\{x_1, x_2, \ldots, x_k\}$, and where each of $\alpha_1, \alpha_2, \ldots, \alpha_q$ is another instruction of one of the forms given here (note that there may be nested while instructions).

The first instruction of ρ is 'INPUT$(x_1, x_2, \ldots, x_l)$' and the last instruction is 'OUTPUT $(x_1, x_2, \ldots, x_l)$', for some l where $1 \leqslant l \leqslant k$. The variables $x_1, x_2, \ldots, x_l$ are the *input-output variables* of ρ, the variables $x_{l+1}, x_{l+2}, \ldots, x_k$ are the *free variables* of ρ and, further, any free variable of ρ never appears on the left-hand side of an assignment instruction nor in a guess instruction. Essentially, free variables appear in ρ as if they were constant symbols.

A program scheme $\rho \in \mathrm{NPSA}(1)$ over σ with s free variables, say, takes a σ-structure $\mathscr{A}$ and s additional values from $|\mathscr{A}|$, one for each free variable of ρ, as input; that is, an expansion $\mathscr{A}'$ of $\mathscr{A}$ by adjoining s additional constants. The program scheme ρ computes on $\mathscr{A}'$ in the obvious way except that:

- execution of the instruction 'GUESS x_i' non-deterministically assigns an element of $|\mathscr{A}|$ to the variable x_i;
- the constants 0 and *max* are interpreted as two arbitrary but distinct elements of $|\mathscr{A}|$; and
- initially, every input–output variable and every array element is assumed to have the value 0.

Note that throughout a computation of ρ, the value of any free variable does not change. The expansion $\mathscr{A}'$ of the structure $\mathscr{A}$ is *accepted* by ρ, and we write $\mathscr{A}' \models \rho$, if, and only if, there exists a computation of ρ on this expansion such that the output-instruction is reached with all input-output variables having the value *max*. (We can easily build the usual 'if' and 'if-then-else' instructions using while instructions: see, for example [38]. Henceforth, we shall assume that these instructions are at our disposal.)

We want the sets of structures accepted by our program schemes to be problems, i.e., closed under isomorphism, and so we only ever consider program schemes ρ where a structure is accepted by ρ when 0 and *max* are given two distinct values from the universe of the structure if, and only if, it is accepted no matter which pair of distinct values is chosen for 0 and *max*. Let us reiterate: when we say that ρ is a program scheme of NPSA(1) we mean that ρ accepts a problem and the acceptance of any input structure does not depend upon the pair of distinct values we give to 0 and *max*. This is analogous to how we build a successor relation or two constant symbols into a logic. Indeed, we can build a successor relation into our program schemes of NPSA(1) so as to obtain the class of program schemes $\mathrm{NPSA}_s(1)$. As with our logics, we write NPSA(1) and $\mathrm{NPSA}_s(1)$ to also denote the class of problems accepted by the program schemes of NPSA(1) and $\mathrm{NPSA}_s(1)$, respectively. It was proven in [38] that a problem is in the complexity class **PSPACE** (polynomial-space) if, and only if, it is in $\mathrm{NPSA}_s(1)$.

Henceforth, we think of our program schemes as being written in the style of a computer program. That is, each instruction is written on one line and while instructions (and, similarly, if and if-then-else instructions) are split so that 'WHILE φ DO' appears on one line, 'α_1' appears on the next, 'α_2' on the next, and so on (of course, if any α_i is a while, if or if-then-else instruction then it is split over a number of lines in the same way). The instructions are labelled 1, 2, and so on, according to the line they appear on. In particular, every instruction is considered to be an assignment, a guess or a test. An *instantaneous description* (*ID*) of a program scheme on some input consists of a value for each variable, the number of the instruction about to be executed and values for all array elements. A *partial ID* consists of just a value for each variable and the number of the instruction about to be executed. One *step* in a program scheme computation is the execution of one instruction, which takes one ID to another, and we

say that a program scheme can *move* from one ID to another if there exists a sequence of steps taking the former ID to the latter.

3. Complete problems

We begin by examining the class of problems NPSA(1) and we show that this class has a complete problem via quantifier-free first-order translations with two constants. This problem, which to our knowledge has not been studied before, is also shown to be complete for **PSPACE** via quantifier-free first-order translations with successor.

Definition 1. Let the signature $\sigma_{TR} = \langle E, P, T, C, D \rangle$, where E is a binary relation symbol, P and T are unary relation symbols and C and D are constant symbols. We can envisage any σ_{TR}-structure $\mathscr{A}$ as a digraph (possibly with self-loops) whose edge relation is E and with distinguished vertices C, the *source*, and D, the *sink*. The relation P can be seen as providing a partition of the vertices and the relation T a subset of the vertices upon which *tokens* are initially placed. All tokens are indistinguishable and any vertex has upon it at most one token. Let us call a σ_{TR}-structure $\mathscr{A}$ a *token digraph*.

Just as one can traverse a path in a digraph by moving along edges, so one can traverse a path in a token digraph $\mathscr{A}$. However, as to how edges can be traversed is different from the usual notion. Consider an edge $(u, v) \in E$ for which both u and v are in P and such that a *traveller* is at vertex u (the traveller traverses a path of edges in the digraph). The edge (u, v) can only be traversed by the traveller moving as follows:

- the traveller moves from u via the edge (u, u') to a vertex u' not in P upon which exactly one token resides;
- then from u' via the edge (u', v') to a vertex v' not in P upon which no token resides, if $v' \neq u'$, and at the same time taking the token previously at u' to v', or by moving from u' via the edge (u', u') (if it exists) to u' (so that the token remains at u'); and finally
- by moving from the vertex v' or u', whichever is the case, via the edge (v', v) or (u', v) to v.

This is called a *compound move* (such a move is illustrated in Fig. 1), and compound moves are the only moves the traveller is allowed to make. Any tokens which happen to initially lie in P are ignored and play no part in any path traversal. Also, the traveller only ever makes compound moves from a vertex of P (u above) to a vertex of P (v above).

The problem *Token Reachability* is defined as all those σ_{TR}-structures, i.e., token digraphs, for which a path can be traversed starting at the source and ending at the sink where the edges are traversed only by compound moves. Any instance for which $C = D$, no matter whether C is in P or not, is a yes-instance (note that if $C \notin P$ then the traveller cannot move).

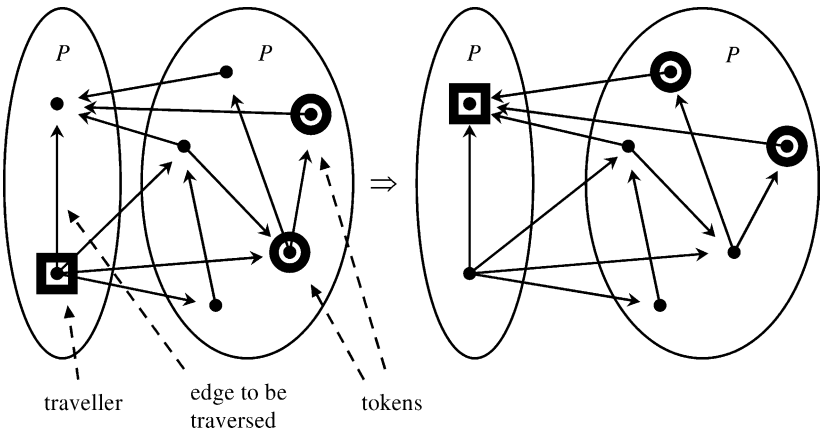

Fig. 1. A compound move in a token digraph.

Theorem 2. *There is a quantifier-free first-order translation with two constants from any problem in* NPSA(1) *to the problem Token Reachability. Hence, Token Reachability is complete for* NPSA(1) *via quantifier-free first-order translations with two constants.*

Proof. Let ρ be a program scheme of NPSA(1) over the signature σ, possibly in which if instructions and if-then-else instructions occur. W.l.o.g. (by introducing more variables if needs be), we may assume that:

- every array symbol only appears in assignment instructions;
- no constant symbol appears in any assignment instruction involving an array symbol; and
- there is only one array symbol B, and this array symbol has dimension $d \geqslant 1$.

Suppose that ρ involves the variables $x_1, x_2, \ldots, x_k$ and that there are l instructions in ρ, numbered $1, 2, \ldots, l$.

Let $\mathscr{A}$ be a σ-structure of size $n \geqslant 2$. An element $\mathbf{u} = (u_0, u_1, \ldots, u_k)$ of $\{1, 2, \ldots, l\} \times |\mathscr{A}|^k$ encodes a partial ID of the program scheme ρ on input $\mathscr{A}$ via: a computation of ρ on $\mathscr{A}$ is about to execute instruction u_0 and the variables $x_1, x_2, \ldots, x_k$ currently have the values $u_1, u_2, \ldots, u_k$, respectively. Henceforth, we identify partial IDs of ρ and the elements of $\{1, 2, \ldots, l\} \times |\mathscr{A}|^k$.

Let the digraph G_0 have vertex set $U_0 = \{1, 2, \ldots, l\} \times |\mathscr{A}|^{k+2}$ and edge set $E_0 = E_0^1 \cup E_0^2 \cup E_0^3$, where E_0^1, E_0^2, and E_0^3 are defined as follows (in our eventual token digraph, the vertices of U_0 will play the role of the vertices of P from Definition 1).

- $E_0^1 = \{((\mathbf{u}, 0, 0), (\mathbf{u}, max, t)), ((\mathbf{u}, max, t), (\mathbf{v}, 0, 0)) \in U_0 \times U_0$: instruction u_0 is of the form $x_j := B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}]$ and it is possible for ρ on input $\mathscr{A}$ to move from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ in one step, and $v_j = t\}$.

- $E_0^2 = \{((\mathbf{u},0,0),(\mathbf{v},0,0)) \in U_0 \times U_0$: instruction u_0 is of the form $B[x_{i_1},x_{i_2},\ldots,x_{i_d}] := x_j$ and it is possible for ρ on input $\mathscr{A}$ to move from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ in one step$\}$.
- $E_0^3 = \{((\mathbf{u},0,0),(\mathbf{v},0,0)) \in U_0 \times U_0$: instruction u_0 does not involve the array symbol B and it is possible for ρ on input $\mathscr{A}$ to move from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ in one step$\}$.

Of course, whether ρ on input $\mathscr{A}$ actually moves from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ in one step at some point in a computation depends upon whether $\mathbf{u}$ can be reached from the initial ID and it might also depend upon the actual value of the array B at that time. The edges of $E_0^1 \cup E_0^2$ reflect *potential* one-step moves from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ (moves which are dependent upon the value of B).

For each $\mathbf{w} \in |\mathscr{A}|^d$, define the digraph $G_{\mathbf{w}}$ to have vertex set $V_{\mathbf{w}} = |\mathscr{A}|$ (all vertex sets of such digraphs are disjoint). The edge set $E_{\mathbf{w}}$ of $G_{\mathbf{w}}$ consists of every possible edge between vertices of $V_{\mathbf{w}}$ including self-loops, i.e., $G_{\mathbf{w}}$ is the complete digraph on vertex set $V_{\mathbf{w}}$.

For each vertex $(\mathbf{u},0,0) \in U_0$, let $H_{\mathbf{u}}$ be a digraph with one vertex $z_{\mathbf{u}}$ and one self-loop $(z_{\mathbf{u}},z_{\mathbf{u}})$ (again, all such digraphs are disjoint).

Let the digraph $\mathscr{G}$ consist of the disjoint union of the digraphs G_0, $\{G_{\mathbf{w}}$: $\mathbf{w} \in |\mathscr{A}|^d\}$ and $\{H_{\mathbf{u}}$: $(\mathbf{u},0,0) \in U_0\}$, together with the following additional edges between the vertices of these digraphs

- If instruction u_0 is of the form $x_j := B[x_{i_1},x_{i_2},\ldots,x_{i_d}]$ then there are edges

  $$\{((\mathbf{u},0,0),z_{\mathbf{u}}),(z_{\mathbf{u}},(\mathbf{u},max,t))\text{: } (\mathbf{u},0,0) \in U_0,\ t \in |\mathscr{A}|\},$$

  and corresponding to every edge $((\mathbf{u},max,t),(\mathbf{v},0,0))$ of E_0^1, there are edges

  $$((\mathbf{u},max,t),t^{\mathbf{w}}) \quad \text{and} \quad (t^{\mathbf{w}},(\mathbf{v},0,0)),$$

  where $\mathbf{w} = (u_{i_1},u_{i_2},\ldots,u_{i_d})$ and $t^{\mathbf{w}}$ is vertex t of $V_{\mathbf{w}}$.
- If instruction u_0 is of the form $B[x_{i_1},x_{i_2},\ldots,x_{i_d}] := x_j$ then corresponding to every edge $((\mathbf{u},0,0),(\mathbf{v},0,0))$ of E_0^2, there are edges

  $$\{((\mathbf{u},0,0),t^{\mathbf{w}}),(u_j^{\mathbf{w}},(\mathbf{v},0,0))\text{: } t \in |\mathscr{A}|\},$$

  where $\mathbf{w} = (u_{i_1},u_{i_2},\ldots,u_{i_d})$ and $t^{\mathbf{w}}$ (resp. $u_j^{\mathbf{w}}$) is vertex t (resp. u_j) of $V_{\mathbf{w}}$.
- If instruction u_0 does not involve the array symbol B then corresponding to every edge $((\mathbf{u},0,0),(\mathbf{v},0,0))$ of E_0^3, there are edges

  $$((\mathbf{u},0,0),z_{\mathbf{u}}) \quad \text{and} \quad (z_{\mathbf{u}},(\mathbf{v},0,0)).$$

That portion of the digraph $\mathscr{G}$ corresponding to a one-step move of ρ on input $\mathscr{A}$ from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ can be visualized as in Figs. 2 and 3 when instruction u_0 is of the form $x_j := B[x_{i_1},x_{i_2},\ldots,x_{i_d}]$ and $B[x_{i_1},x_{i_2},\ldots,x_{i_d}] := x_j$, respectively.

We can now extend $\mathscr{G}$ to a token digraph: let the source be the vertex $(1,\mathbf{0},0,0)$ of U_0 and the sink be the vertex $(l,\mathbf{max},0,0)$ of U_0; let $P = U_0$; and let $T = \{0^{\mathbf{w}}$: $0^{\mathbf{w}}$ is the vertex 0 of $V_{\mathbf{w}}$, where $\mathbf{w} \in |\mathscr{A}|^d\} \cup \{z_{\mathbf{u}}$: $(\mathbf{u},0,0) \in U_0\}$.

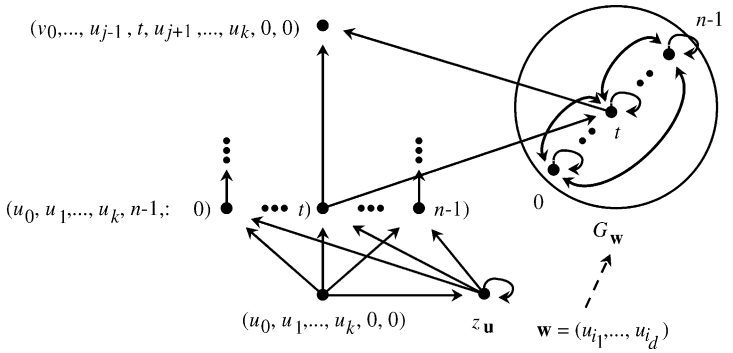

Fig. 2. A portion of the digraph $\mathscr{G}$ corresponding to an instruction of the form $x_j := B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}]$.

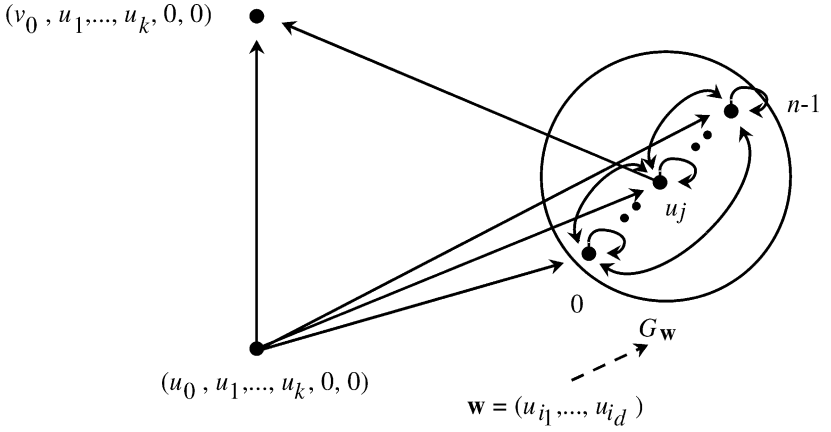

Fig. 3. A portion of the digraph $\mathscr{G}$ corresponding to an instruction of the form $B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}] := x_j$.

Suppose that $\mathscr{A}$ is accepted by the program scheme ρ. Then there is a sequence s of (full, not partial) IDs starting at the initial ID (where all variables have the value 0, where the instruction to be executed is instruction 1 and where the array B has the value 0 throughout) and ending in a final ID (where all variables have the value *max* and where the instruction to be executed is instruction l) such that ρ moves from one ID in s to the next in one step. We can mirror any ID in a configuration of our token digraph $\mathscr{G}$ as follows. If the ID consists of the partial ID $\mathbf{u} \in \{1, 2, \ldots, l\} \times |\mathscr{A}|^k$ together with some valuation on the array B then we can position our traveller at node $(\mathbf{u}, 0, 0)$ of U_0 (the traveller marks the progress in a path traversal). We place a token on vertex $v^{\mathbf{w}}$ of $V_{\mathbf{w}}$ if, and only if, the current value of $B[\mathbf{w}]$ is v, where $v \in |\mathscr{A}|$ and $\mathbf{w} \in |\mathscr{A}|^d$, and we put a token on each vertex of $\{z_{\mathbf{u}}: (\mathbf{u}, 0, 0) \in U_0\}$. Note that the initial ID of ρ on input $\mathscr{A}$ is mirrored by the initial configuration of $\mathscr{G}$.

Suppose we are at some ID, I say, in the sequence s, mirrored by some configuration in $\mathscr{G}$ as described in the preceding paragraph, and the next ID in s is J. In particular, let the partial ID associated with ID I be $\mathbf{u}$.

- If the instruction of ρ which takes ID I to ID J is of the form $x_j := B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}]$ then consider the following compound moves for the traveller in the token digraph $\mathscr{G}$. Suppose that $B[\mathbf{w}] = t$ in ID I, where $\mathbf{w} = (u_{i_1}, u_{i_2}, \ldots, u_{i_d})$.
  - First compound move: $(\mathbf{u}, 0, 0) \to z_{\mathbf{u}} \to z_{\mathbf{u}} \to (\mathbf{u}, max, t)$.
  - Second compound move: $(\mathbf{u}, max, t) \to t^{\mathbf{w}} \to t^{\mathbf{w}} \to (\mathbf{v}, 0, 0)$, where $\mathbf{v}$ is the partial ID associated with ID J.

  These are valid compound moves and after the moves, no token has changed position.
- If the instruction of ρ which takes ID I to ID J is of the form $B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}] := x_j$ then consider the following compound move for the traveller in the token digraph $\mathscr{G}$. Suppose that $B[\mathbf{w}] = t$ in ID I, where $\mathbf{w} = (u_{i_1}, u_{i_2}, \ldots, u_{i_d})$.
  - Compound move: $(\mathbf{u}, 0, 0) \to t^{\mathbf{w}} \to u_j^{\mathbf{w}} \to (\mathbf{v}, 0, 0)$, where $\mathbf{v}$ is the partial ID associated with ID J.

  This is a valid compound move and after the move, the token in $V_{\mathbf{w}}$ has moved from vertex $t^{\mathbf{w}}$ to vertex $u_j^{\mathbf{w}}$ (which might be the same vertex).
- If the instruction of ρ which takes ID I to ID J does not involve the array symbol B then consider the following compound move for the traveller in the token digraph $\mathscr{G}$.
  - Compound move: $(\mathbf{u}, 0, 0) \to z_{\mathbf{u}} \to z_{\mathbf{u}} \to (\mathbf{v}, 0, 0)$, where $\mathbf{v}$ is the partial ID associated with ID J.

  This is a valid compound move and after the move, no token has changed position.

Note that the resulting configuration in $\mathscr{G}$ mirrors the ID J. Consequently, the token digraph $\mathscr{G}$ is a yes-instance of the problem *Token Reachability*.

Conversely, suppose that $\mathscr{G}$ is a yes-instance of the problem *Token Reachability*. Then there is a sequence s of configurations of $\mathscr{G}$ obtained by continually making compound moves, starting at the initial configuration, such that in the final configuration of s the traveller is at the sink $(l, \mathbf{max}, 0, 0)$. Every configuration of s is such that:

- for every $\mathbf{w} \in |\mathscr{A}|^d$, there is exactly one token on the vertices of $V_{\mathbf{w}}$; and
- there is a token on every vertex of $\{z_{\mathbf{u}}: (\mathbf{u}, 0, 0) \in U_0\}$.

Suppose that the traveller is at the vertex $(\mathbf{u}, 0, 0)$ of U_0 in one of these configurations. Then this configuration mirrors an ID I of ρ on input $\mathscr{A}$, as above.

In order to get to the next configuration in s, the traveller must make a compound move from this configuration. There are three possibilities.

- If instruction u_0 in ρ is of the form $x_j := B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}]$ then the traveller's next two compound moves must be:
  - a compound move to a vertex $(\mathbf{u}, max, t)$ of U_0, for the unique $t \in |\mathscr{A}|$ for which there is a token on vertex $t^{\mathbf{w}}$ of $V_{\mathbf{w}}$, where $\mathbf{w} = (u_{i_1}, u_{i_2}, \ldots, u_{i_d})$;
  - followed by a compound move from vertex $(\mathbf{u}, max, t)$ of U_0 to some vertex $(\mathbf{v}, 0, 0)$ of U_0, where $\mathbf{v} = (v_0, u_0, \ldots, u_{j-1}, t, u_{j+1}, \ldots, u_k)$ and where it is possible for ρ on input $\mathscr{A}$ to move from the partial ID $\mathbf{u}$ to the partial ID $\mathbf{v}$ in one step.

  Note that in making these compound moves, no token has changed position.

- If instruction u_0 in ρ is of the form $B[x_{i_1}, x_{i_2}, \ldots, x_{i_d}] := x_j$ then the traveller's next compound move must be to some vertex $(\mathbf{v}, 0, 0)$ of U_0, where it is possible for ρ on input $\mathscr{A}$ to move from the partial ID $\mathbf{u}$ to the partial ID $\mathbf{v}$ in one step. In making this compound move, the token corresponding to $V_{\mathbf{w}}$, where $\mathbf{w} = (u_{i_1}, u_{i_2}, \ldots, u_{i_d})$, is moved to the vertex u_j of $V_{\mathbf{w}}$.
- If instruction u_0 in ρ does not involve the array symbol B then the traveller's next compound move must be to a vertex $(\mathbf{v}, 0, 0)$ of U_0, where it is possible for ρ on input $\mathscr{A}$ to move from partial ID $\mathbf{u}$ to partial ID $\mathbf{v}$ in one step.

No matter which of these three possibilities occurs, the resulting configuration mirrors an ID J such that ρ on input $\mathscr{A}$ can move from ID I to ID J in one step. Consequently, if the token digraph $\mathscr{G}$ is a yes-instance of *Token Reachability* then ρ accepts $\mathscr{A}$.

The token digraph $\mathscr{G}$ can easily be described in terms of $\mathscr{A}$ by a quantifier-free first-order formula with two constants (see comparable constructions in, for example, [39]), and so the problem *Token Reachability* is hard for NPSA(1) via quantifier-free first-order translations with two constants. Moreover, *Token Reachability* can be accepted by the following program scheme of NPSA(1).

```
input(u, v, u', v', w)
guess w
while w = max do
  guess w
  if T(w) then B[w] := max fi
  guess w
  od
if ¬P(C) ∧ C ≠ D then 'loop forever' fi
u := C
while u ≠ D do
  guess v
  if ¬P(v) ∨ ¬E(u, v) then 'loop forever' fi
  guess u'
  if P(u') ∨ ¬E(u, u') ∨ B[u'] = 0 then 'loop forever' fi
  guess v'
  if P(v') ∨ ¬E(u', v') ∨ (B[v'] = max ∧ u' ≠ v')
                        ∨(T(v') ∧ M[v'] = 0 ∧ u' ≠ v') then
    'loop forever'
    fi
  if E(v', v) then
    B[u'] := 0; B[v'] := max; M[u'] := max; u := v
  else
    'loop forever'
    fi
  od
(u, v, u', v', w) := (max, max, max, max, max)
output(u, v, u', v', w)
```

Some explanation is in order (beyond the obvious short-hand we use in our description). Our program scheme ρ involves two array symbols, B and M, both of dimension 1. Suppose that some σ_{TR}-structure $\mathscr{A}$ is accepted by ρ. The first part of ρ guesses a set of vertices upon which tokens initially lie: this set is $\{w\colon B[w] = max\}$. Throughout the execution, the array B details the locations of these tokens as they are moved about: call these tokens the B-type tokens. No other token is ever moved (and by 'moved' we include tokens which are moved along self-loops). The array M is initially identically 0 but whenever a vertex w upon which a B-type token lies is involved in a compound move, the array element $M[w]$ is set at max. Consequently, at any particular time we know where the B-type tokens are (the elements w for which $B[w] = max$) and we know where the other tokens are (the elements w for which $T(w) \wedge M[w] = 0$).

The code within the second while-loop is the code associated with making a compound move. The variable v holds the vertex $v \in P$ (see Definition 1), the variable u' holds the vertex $u' \notin P$ and the variable v' holds the vertex $v' \notin P$ (or possibly u'). Note that in choosing u', we must ensure that a B-type token currently lies on u' (as these are the only tokens which we are allowed to move). This is done by checking to see whether $B[u'] = max$. Also, in choosing v' (if it is to be different from u') we must ensure that no B-type token lies on v' nor no non-B-type token. Thus, $\mathscr{A}$ is in *Token Reachability*.

Conversely, suppose that $\mathscr{A}$ is in *Token Reachability*. In the initial phase of the execution of ρ on $\mathscr{A}$, we can guess every token to be a B-type token. The result follows. □

Theorem 2 reinforces any notions we might have had that NPSA(1) is a class of problems worthy of study in that it provides a complete problem for this class via a reasonable notion of reduction (recall, **NP** (non-deterministic polynomial-time) has complete problems via quantifier-free first-order translations with two constants [10, 37]).

Corollary 3. *Token Reachability is complete for* **PSPACE** *via quantifier-free first-order translations with successor.*

Proof. By [38], any problem in **PSPACE** can be accepted by some program scheme of NPSA$_s$(1). Hence, the result follows from Theorem 2. □

Note that if we were to *define* **PSPACE** as consisting of all those problems accepted by program schemes of NPSA$_s$ then we would have proven that *Token Reachability* was complete for **PSPACE** without relying on the existence of any other **PSPACE**-complete problem; that is, *Token Reachability* would have been proven complete from first principles. Of course, this is not how **PSPACE** was originally defined; however, the move from the Turing-machine-based definition of **PSPACE** to the program-scheme-based one is not particularly involved.

We now exhibit another complete problem for NPSA(1), via quantifier-free first-order translations with two constants. Our reasons for doing so are because the particular problem involves Petri nets, and Petri nets form a fundamental model in the study of concurrency, and because the computational complexity of the particular problem has been studied before. We refer the reader to [14] for the basic notions and definitions concerning Petri nets and for a survey of complexity-theoretic results involving Petri nets.

Definition 4. Let the signature $\sigma_{PN} = \langle T_1, T_2, M, C \rangle$, where M is a unary relation symbol, T_1 is a binary relation symbol, T_2 is a relation symbol of arity 4 and C is a constant symbol. We can envisage a σ_{PN}-structure $\mathscr{A}$ as a Petri net whose places are given by $|\mathscr{A}|$ and whose transitions are given by T_1 and T_2 via:

- there is a transition $(\{x\}, \{y\})$ whose input place is $\{x\}$ and whose output place is $\{y\}$ if, and only if, $T_1(x, y)$ holds; and
- there is a transition $(\{x_1, x_2\}, \{y_1, y_2\})$ whose input places are $\{x_1, x_2\}$ and whose output places are $\{y_1, y_2\}$ if, and only if, $T_2(x_1, x_2, y_1, y_2)$ holds, where $x_1 \neq x_2$ and $y_1 \neq y_2$.

The relation M can be seen as providing an initial marking (with one token on place p if, and only if, $M(p)$ holds) and the constant C as providing a final marking (consisting of one token on the place C).

A σ_{PN}-structure $\mathscr{A}$, i.e., a Petri net, complete with inital and final markings, where every transition has either 2 input places and 2 output places or 1 input place and 1 output place, is in the problem *Petri Net Coverability* if, and only if, there is a marking covering the final marking that is reachable from the initial marking, i.e., there is a reachable marking in which there is at least one token on the place C. □

Note that our problem *Petri Net Coverability* is *not* the coverability problem for general Petri nets, where transitions might have any number of input places and output places, where more than one token might be on a place in the initial or final marking, and where the final marking need not consist of just one token on one place. In our problem, we are considering a very restricted class of Petri nets. The coverability problem for general Petri nets, which is recursively equivalent to the *reachability problem* for general Petri nets [23], was shown to be decidable by Mayr [32], Lipton [31] having previously proved an exponential space lower bound for the reachability problem. Our problem, namely *Petri Net Coverability*, was (essentially) shown to be complete for **PSPACE** via logspace reductions by Jones et al. [27].

Theorem 5. *There is a quantifier-free first-order translation with two constants from any problem in* NPSA(1) *to the problem Petri Net Coverability. Hence, Petri Net Coverability is complete for* NPSA(1) *via quantifier-free first-order translations with two constants.*

Proof. Let us proceed within the context of the proof of Theorem 2. The places of our Petri net $\mathscr{P}$, built from the structure $\mathscr{A}$, are the vertices of the digraph $\mathscr{G}$ of the

proof of Theorem 2. Replace:

- every edge $((\mathbf{u},0,0),(\mathbf{u},max,t))$ of the subset of edges E_0^1 of $\mathscr{G}$ with the transition $(\{(\mathbf{u},0,0)\},\{(\mathbf{u},max,t)\})$;
- every edge $((\mathbf{u},max,t),(\mathbf{v},0,0))$ of the subset of edges E_0^1 of $\mathscr{G}$ with the transition $(\{(\mathbf{u},max,t),t^{\mathbf{w}}\},\{t^{\mathbf{w}},(\mathbf{v},0,0)\})$;
- every edge $((\mathbf{u},0,0),(\mathbf{v},0,0))$ of the subset of edges E_0^2 of $\mathscr{G}$ with the transitions $\{(\{(\mathbf{u},0,0),t^{\mathbf{w}})\},\{u_j^{\mathbf{w}},(\mathbf{v},0,0)\}): t\in|\mathscr{A}|\}$; and
- every edge $((\mathbf{u},0,0),(\mathbf{v},0,0))$ of the subset of edges E_0^3 of $\mathscr{G}$ with the transition $(\{(\mathbf{u},0,0)\},\{(\mathbf{v},0,0)\})$.

Our initial marking M consists of one token on the place $(1,0,\ldots,0)\in U_0$ and the places in $\{0^{\mathbf{w}}\in V_{\mathbf{w}}\colon \mathbf{w}\in|\mathscr{A}|^d\}$. The final marking consists of one token on the place $(l,max,\ldots,max)\in U_0$. By reasoning similarly to as in the proof of Theorem 2, we obtain that $\mathscr{A}\models\rho$ if, and only if, $\mathscr{P}\in$ *Petri Net Coverability*.

Our Petri net $\mathscr{P}$ can easily be described in terms of $\mathscr{A}$ by quantifier-free first-order formula with two constants, and so the problem *Petri Net Coverability* is hard for NPSA(1) via quantifier-free first-order translations with two constants. Moreover, *Petri Net Coverability* can be accepted by some program scheme of NPSA(1) as we now explain. Note that any reachable marking in one of our Petri nets $\mathscr{P}$ might be such that a place has upon it more than one token (even though the initial and final markings have only at most one token on each place), but never more than n tokens. We can encode the number of tokens on each place in $\mathscr{P}$ as follows. We use two dedicated array symbols B_1 and B_2, both of dimension 2. If the place p has no tokens upon it then

$$B_1[p,p]=0 \quad \text{and} \quad B_2[p,p]=0.$$

If the place p has $t>0$ tokens upon it then

$$B_1[p,p]=p_1,\quad B_1[p_1,q_1]=p_2,\quad B_1[p_2,q_2]=p_3,\ldots,B_1[p_t,q_t]=0,$$

$$B_2[p,p]=q_1,\quad B_2[p_1,q_1]=q_2,\quad B_2[p_2,q_2]=q_3,\ldots,B_2[p_t,q_t]=0$$

for some distinct pairs $(p,p),(p_1,q_1),(p_2,q_2),\ldots,(p_t,q_t)\in|\mathscr{P}|^2$. It is straightforward to write a 'fragment of code' to increment the number of tokens upon some place or to decrement the number of tokens on some place, or to check whether there is at least one token on some place. We keep track of the movement of tokens in an execution of our Petri net as we did in the program scheme accepting the problem *Token Reachability* in the proof of Theorem 2 (where we used the array symbols B and M). Hence, we can clearly derive a program scheme of NPSA(1) to accept the problem *Petri Net Coverability*. □

The following corollary follows from Theorem 5 just as Corollary 3 follows from Theorem 2.

Corollary 6. *Petri Net Coverability is complete for* **PSPACE** *via quantifier-free first-order translations with successor.*

4. Logics and program schemes

In this section, we extend the program schemes of NPSA(1) so that we obtain a class of program schemes NPSA with the property that a problem can be accepted by a program scheme of NPSA if, and only if, it can be defined by a sentence of the logic $(\pm\mathrm{TR})^*[\mathrm{FO}]$, where TR is the operator corresponding to the problem *Token Reachability*.

An important point to note is that whereas the usual existential quantifier is catered for in program schemes of NPSA(1) via the guess instruction (intuitively speaking), there is no such analogous modelling of the universal quantifier. Consequently, we extend our program schemes by introducing universal quantification in the following manner.

Definition 7. Let σ be some signature. For some $m \geqslant 1$, let the program scheme $\rho \in \mathrm{NPSA}(2m-1)$ be over the signature σ and involve the variables $x_1, x_2, \ldots, x_k$. Suppose that the variables $x_1, x_2, \ldots, x_l$ are the input-output variables of ρ, the variables $x_{l+1}, x_{l+2}, \ldots, x_{l+s}$ are the free variables, and the remaining variables are the bound variables (note that if $m = 1$ then ρ has no bound variables but that this may not be the case if $m > 1$). Let $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$ be free variables of ρ, for some p such that $1 \leqslant p \leqslant s$. Then

$$\forall x_{i_1} \forall x_{i_2} \ldots \forall x_{i_p} \rho$$

is a program scheme of NPSA($2m$), which we denote by ρ', with no input-output variables, with free variables those of $\{x_{l+1}, x_{l+2}, \ldots, x_{l+s}\} \setminus \{x_{i_1}, x_{i_2}, \ldots, x_{i_p}\}$ and with the remaining variables of $\{x_1, x_2, \ldots, x_k\}$ as its bound variables.

A program scheme such as ρ' takes expansions $\mathscr{A}'$ of σ-structures $\mathscr{A}$ by adjoining $s - p$ constants as input (one for each free variable), and ρ' accepts such an expansion $\mathscr{A}'$ if, and only if, for every expansion $\mathscr{A}''$ of $\mathscr{A}'$ by p additional constants (one for each variable x_{i_j}), $\mathscr{A}'' \models \rho$.

Note that the different computations of ρ on expansions $\mathscr{A}''$ of $\mathscr{A}'$, in Definition 7, are all such that all arrays are initialised to 0.

Definition 8. Let σ be some signature. A program scheme $\rho' \in \mathrm{NPSA}(2m-1)$, for some $m \geqslant 2$, over the signature σ and involving the variables of $\{x_1, x_2, \ldots, x_k\}$, is formed exactly as are the program schemes of NPSA(1), with the input-output and free variables defined accordingly, except that the test in some while instruction is a program scheme $\rho \in \mathrm{NPSA}(2m-2)$ whose free and bound variables are all from

$\{x_1, x_2, \ldots, x_k\}$ (note that ρ has no input-output variables). However, there are further stipulations:

- all free variables in any test $\rho \in \mathrm{NPSA}(2m-2)$ in any while instruction are input-output or free variables of ρ';
- the bound variables of ρ' consist of all bound variables of any test $\rho \in \mathrm{NPSA}(2m-2)$ in any while instruction (and no bound variable is ever an input-output or free variable of ρ); and
- this accounts for all variables of $\{x_1, x_2, \ldots, x_k\}$.

Of course, any free variable of ρ' never appears on the left-hand side of an assignment instruction or in a guess instruction.

A program scheme $\rho' \in \mathrm{NPSA}(2m-1)$ takes expansions $\mathscr{A}'$ of σ-structures $\mathscr{A}$ by adjoining s constants as input, where s is the number of free variables, and computes on $\mathscr{A}'$ in the obvious way except that when some while instruction is encountered, the test, which is a program scheme $\rho \in \mathrm{NPSA}(2m-2)$, is evaluated according to the expansion of $\mathscr{A}'$ by the current values of any relevant input-output variables of ρ' (which may be free in ρ). In order to evaluate this test, all arrays in ρ are initialised to 0 and when the test has been evaluated the computation of ρ' resumes accordingly with its arrays having exactly the same values as they had immediately before the test was evaluated.

In a program scheme such as ρ' in Definition 8, the only information which can be 'passed' to a test evaluation is the current values of the relevant input-output or free variables. Arrays cannot be used to pass information across. As we mentioned in the Introduction, if our semantics were such as to allow universal quantification over arrays then we could build our own successor relation.

Remark 9. A simple analysis yields that we can build the usual 'if' and 'if-then-else' instructions in the program schemes of NPSA(m), for all odd $m \geqslant 1$. Indeed, henceforth we assume that these instructions are at our disposal.

Theorem 2 allows us to relate the class of problems accepted by the program schemes of NPSA with the class of problems defined by the sentences of the logic $(\pm\mathrm{TR})^*[\mathrm{FO}]$. For each $m \geqslant 1$, we define the fragment $\pm\mathrm{TR}(m)$ of $(\pm\mathrm{TR})^*[\mathrm{FO}]$ as follows (see [2] for similarly defined fragments of other logics).

- $\pm\mathrm{TR}(1)$ consists of all formulae of the form $\mathrm{TR}[\lambda\mathbf{x}, \mathbf{y}\psi_E, \mathbf{x}\psi_P, \mathbf{x}\psi_T](\mathbf{u}, \mathbf{v})$, where ψ_E, ψ_P and ψ_T are quantifier-free first-order formulae and where $\mathbf{u}$ and $\mathbf{v}$ are tuples of constant symbols or variables.
- $\pm\mathrm{TR}(m+1)$, for odd $m \geqslant 1$, consists of the universal closure of $\pm\mathrm{TR}(m)$; that is, the set of formulae of the form $\forall z_1 \forall z_2 \ldots \forall z_k \psi$, where ψ is a formula of $\pm\mathrm{TR}(m)$.
- $\pm\mathrm{TR}(m+1)$, for even $m \geqslant 2$, consists of the set of formulae of the form

$$\mathrm{TR}[\lambda\mathbf{x}, \mathbf{y}(\psi_E^1 \vee \neg\psi_E^2), \mathbf{x}(\psi_P^1 \vee \neg\psi_P^2), \mathbf{y}(\psi_T^1 \vee \neg\psi_T^2)](\mathbf{u}, \mathbf{v}),$$

where ψ_E^1, ψ_E^2, ψ_P^1, ψ_P^2, ψ_T^1 and ψ_T^2 are formulae of $\pm\mathrm{TR}(m)$ and where $\mathbf{u}$ and $\mathbf{v}$ are tuples of constant symbols or variables.

A straightforward induction yields that:

- for every odd $m \geqslant 1$, every formula in the closure of $\pm\mathrm{TR}(m)$ under $\wedge$, $\vee$ and $\exists$ is logically equivalent to a formula of $\pm\mathrm{TR}(m)$; and
- for every even $m \geqslant 1$, every formula in the closure of $\pm\mathrm{TR}(m)$ under $\wedge$, $\vee$ and $\forall$ is logically equivalent to a formula of $\pm\mathrm{TR}(m)$.

Consequently, $(\pm\mathrm{TR})^*[\mathrm{FO}] = \bigcup\{\mathrm{TR}(m)\colon m \geqslant 1\}$.

Corollary 10. *In the presence of two built-in constant symbols,* $\pm\mathrm{TR}(m) = \mathrm{NPSA}(m)$, *for each* $m \geqslant 1$; *and so* $(\pm\mathrm{TR})^*[\mathrm{FO}] = \mathrm{NPSA}$.

Proof. We proceed by induction on m. The base case, when $m = 1$, follows by Theorem 2. Suppose, as our induction hypothesis, that $\mathrm{NPSA}(i) = \pm\mathrm{TR}(i)$, for all $i < m$. There are two cases: when m is odd and when m is even.

Suppose that $m \geqslant 2$ is even and let Ω be some problem in $\mathrm{NPSA}(m)$. By the induction hypothesis, Ω can be defined by a sentence of the form

$$\forall z_1 \forall z_2 \ldots \forall z_k \beta(z_1, z_2, \ldots, z_k),$$

where β is a formula of $\pm\mathrm{TR}(m-1)$; that is, $\Omega \in \pm\mathrm{TR}(m)$.

Suppose that $m \geqslant 3$ is odd and let Ω be some problem over σ accepted by a program scheme ρ in $\mathrm{NPSA}(m)$. Replace every occurrence in ρ of a program scheme ρ' of $\mathrm{NPSA}(m-1)$, whose free variables are those of the tuple $\mathbf{y}$, by the atomic formula $R_{\rho'}(\mathbf{y})$, where $R_{\rho'}$ is a new relation symbol corresponding to ρ' (w.l.o.g. we may assume that every such program scheme ρ' has at least one free variable): so in general lots of new relation symbols will be introduced. The resulting program scheme ρ'' is over the signature σ extended with these new relation symbols and is a program scheme of NPSA(1). Note that in any instruction of ρ'' there is at most one occurrence of any of the new relation symbols. Now apply the proof of Theorem 2 to encode a computation of ρ'' as a sentence of $\mathrm{TR}^1[\mathrm{FO}]$. This sentence is of the form

$$\mathrm{TR}[\lambda\mathbf{x}, \mathbf{y}\psi_E, \mathbf{x}\psi_P, \mathbf{x}\psi_T](\mathbf{0}, \mathbf{max}),$$

where ψ_E is of the form

$$\bigvee_{i\in I}(\alpha_i \wedge \beta_i) \vee \bigvee_{j\in J}(\gamma_j \wedge \neg\delta_j)$$

for some finite sets I and J, where each α_i and γ_j is a conjunction of atomic and negated atomic formulae and each β_i and δ_j is a $\mathrm{NPSA}(m-1)$ predicate, i.e., one of the new relation atoms, and where ψ_P and ψ_T are quantifier-free first-order. Essentially, ψ_E has a conjunction of atoms and negated atoms for each possible flow of control of ρ'' from one instruction to another. The β_i's represent the flow of control if an if or a while instruction evaluates to true and the δ_j's if it evaluates to false; and the α_i's

and the γ_j's encode the names of the instructions involved. However, by the induction hypothesis and the remarks immediately preceding the statement of this result,

$$\bigvee_{i\in I}(\alpha_i \wedge \beta_i) \text{ is a NSPA } (m-1) \text{ predicate,}$$

as is

$$\bigwedge_{j\in J}(\neg\gamma_j \vee \delta_j) = \neg \bigvee_{j\in J}(\gamma_j \wedge \neg\delta_j).$$

Consequently, the induction hypothesis yields that $\Omega \in \pm\mathrm{TR}(m)$.

A straightforward induction yields that $\pm\mathrm{TR}(m) \subseteq \mathrm{NPSA}(m)$, and so the result follows. □

In exactly the same way we can obtain the following result.

Corollary 11. *In the presence of* 2 *built-in constant symbols*, $\pm\mathrm{PN}(m) = \mathrm{NPSA}(m)$, *for each* $m \geqslant 1$; *and so* $(\pm\mathrm{PN})^*[\mathrm{FO}] = \mathrm{NPSA}$.

The upshot is that we have equated the class of program schemes NPSA with extensions of first-order logic using uniform sequence of Lindström quantifiers (corresponding to a **PSPACE**-complete problem in both cases).

The following is immediate from Corollaries 3, 6, 10 and 11.

Corollary 12. *In the presence of a built-in successor relation*, $(\pm\mathrm{TR})^*[\mathrm{FO}_s] = \mathrm{TR}^1[\mathrm{FO}_s] = (\pm\mathrm{PN})^*[\mathrm{FO}_s] = \mathrm{PN}^1[\mathrm{FO}_s] = \mathbf{PSPACE}$.

Let us now focus on a comparison of these classes with other classes of logically-defined problems. Bounded-variable infinitary logic plays a prominent role in finite model theory. For any $k \geqslant 1$, $\mathscr{L}^k_{\infty\omega}$ consists of those formulae built using the usual first-order constructs except that infinite conjunctions and disjunctions are allowed but only the variables $x_1, x_2, \ldots, x_k$ may be used. *Bounded-variable infinitary logic*, $\mathscr{L}^\omega_{\infty\omega}$, is defined as $\bigcup\{\mathscr{L}^k_{\infty\omega}: k \geqslant 1\}$ (see [13] for more details). The logic $\mathscr{L}^\omega_{\infty\omega}$ subsumes many logics from finite model theory, notably transitive closure logic, path system logic, least-fixed point logic and partial-fixed point logic (again, see [13]).

Let $\sigma_2 = \langle E\rangle$, where E is a binary relation symbol. We can think of a σ_2-structure $\mathscr{A}$ as an undirected graph via 'there is an edge (u,v) if, and only if, $u \neq v \wedge (E(u,v) \vee E(v,u))$ holds in $\mathscr{A}$'. Define the problem CUB as

$$\mathrm{CUB} = \{\mathscr{A} \in \mathrm{STRUCT}(\sigma_2): \text{ the graph } \mathscr{A} \text{ has a subset of edges inducing a regular subgraph of degree } 3\},$$

where the subgraph induced by a set of edges F of a graph is that subgraph whose vertex set consists of all those vertices incident with at least one edge of F and whose edge set consists of F. The problem CUB has long been known to be **NP**-complete via

logspace reductions (a result attributed to Chvàtal in [17]); and has even been shown to be complete for **NP** via quantifier-free first-order translations with successor when restricted to the class of connected bipartite graphs of maximal degree at most 4 [41]. Of concern to us is the result from [42] that the problem CUB is not definable in $\mathscr{L}^{\omega}_{\infty\omega}$ (even when we allow *counting quantifiers* in $\mathscr{L}^{\omega}_{\infty\omega}$: see [42]).

Proposition 13. *Any problem definable by a sentence of the form*

$$\mathrm{CUB}[\lambda \mathbf{x}, \mathbf{y}\psi(\mathbf{x}, \mathbf{y})],$$

where $|\mathbf{x}| = |\mathbf{y}| = k$, *for some* k, *and* ψ *is a quantifier-free first-order formula with two constants, can be accepted by a program scheme of* NPSA(1).

Proof. We assume throughout that $k = 2$: the general case is similar. Let $\mathscr{A}$ be some structure, of size n, over the underlying signature σ. Our program scheme $\rho \in \mathrm{NPSA}(1)$ proceeds as follows. We begin by 'guessing' a set of (at most $n^2(n^2-1)/2$) distinct potential edges in the graph $\mathscr{G}_{\mathscr{A}}$ described by ψ (interpreted in $\mathscr{A}$). We use the 4-dimensional arrays A_1, A_2, B_1 and B_2 in order to store the guessed list of edges as follows. We guess elements u_1^1, u_2^1, v_1^1 and v_2^1 of $|\mathscr{A}|$, ensuring that it is not the case that all of these elements are equal to 0, and we set

$$A_1[0,0,0,0] := u_1^1, \quad A_2[0,0,0,0] := u_2^1, \quad B_1[0,0,0,0] := v_1^1, \quad B_2[0,0,0,0] := v_2^1.$$

Next, we guess elements u_1^2, u_2^2, v_1^2 and v_2^2 of $|\mathscr{A}|$, and check that $(u_1^2, u_2^2, v_1^2, v_2^2)$ is different from $(0,0,0,0)$ and $(u_1^1, u_2^1, v_1^1, v_2^1)$. If so then we set

$$\begin{aligned} A_1[u_1^1, u_2^1, v_1^1, v_2^1] &:= u_1^2, \quad A_2[u_1^1, u_2^1, v_1^1, v_2^1] := u_2^2, \\ B_1[u_1^1, u_2^1, v_1^1, v_2^1] &:= v_1^2, \quad B_2[u_1^1, u_2^1, v_1^1, v_2^1] := v_2^2. \end{aligned}$$

We stop if each of u_1^2, u_2^2, v_1^2 and v_2^2 is equal to *max*. Next, we guess elements u_1^3, u_2^3, v_1^3 and v_2^3 of $|\mathscr{A}|$ and check that $(u_1^3, u_2^3, v_1^3, v_2^3)$ is different from $(0,0,0,0)$, $(u_1^1, u_2^1, v_1^1, v_2^1)$ and $(u_1^2, u_2^2, v_1^2, v_2^2)$. If so then we set

$$\begin{aligned} A_1[u_1^2, u_2^2, v_1^2, v_2^2] &:= u_1^3, \quad A_2[u_1^2, u_2^2, v_1^2, v_2^2] := u_2^3, \\ B_1[u_1^2, u_2^2, v_1^2, v_2^2] &:= v_1^3, \quad B_2[u_1^2, u_2^2, v_1^2, v_2^2] := v_2^3. \end{aligned}$$

We stop if each of u_1^3, u_2^3, v_1^3 and v_2^3 is equal to *max*; and so on.

Any computation of ρ which completes this first phase is such that the arrays now encode the (non-empty) list of $m-1$ distinct 'potential edges'

$$((u_1^1, u_2^1), (v_1^1, v_2^1)), ((u_1^2, u_2^2), (v_1^2, v_2^2)), \ldots, ((u_1^{m-1}, u_2^{m-1}), (v_1^{m-1}, v_2^{m-1})).$$

We now check that the potential edges on this list are indeed edges of $\mathscr{G}_{\mathscr{A}}$ by verifying that

$$(u_1^i \neq v_1^i \vee u_2^i \neq v_2^i) \wedge (\psi(u_1^i, u_2^i, v_1^i, v_2^i) \vee \psi(v_1^i, v_2^i, u_1^i, u_2^i))$$

holds in $\mathscr{A}$, for $i = 1, 2, \ldots, m-1$.

Finally, we check that each vertex incident with some edge in our list of edges is incident with exactly 3 such edges: if so, we accept the input structure $\mathscr{A}$ otherwise we reject it. It is clear that all of the above can be implemented in a program scheme of NPSA(1); and that the resulting program scheme accepts exactly the problem defined by the sentence CUB$[\lambda \mathbf{x}, \mathbf{y}\psi(\mathbf{x}, \mathbf{y})]$. □

Using the facts that the problem CUB is not definable in $\mathscr{L}^{\omega}_{\infty\omega}$ and that there are non-recursive problems which are definable in $\mathscr{L}^{\omega}_{\infty\omega}$, we immediately obtain the following corollary.

Corollary 14. *There are problems in* NPSA(1) *which are not definable in* $\mathscr{L}^{\omega}_{\infty\omega}$, *and there are problems in* $\mathscr{L}^{\omega}_{\infty\omega}$ *which are not definable in* NPSA(1).

Note that whilst we know that there are problems in NPSA which are not definable in partial fixed-point logic (a fragment of bounded-variable infinitary logic, remember), we do not as yet know whether there are problems in partial fixed-point logic which are not definable in NPSA (although we suspect that there are).

5. Zero-one laws

In this section we prove that the logic NPSA has a zero-one law. Note that by Corollary 14, there is no chance of proving this by using the fact that bounded-variable infinitary logic has a zero-one law [29].

Let σ be a relational signature and let Ω be a problem over σ. Define the fraction

$$l_n(\Omega) = \frac{|\{\mathscr{A}\colon\ \mathscr{A} \in \mathrm{STRUCT}(\sigma) \text{ has size } n \text{ and } \mathscr{A} \in \Omega\}|}{|\{\mathscr{A}\colon\ \mathscr{A} \in \mathrm{STRUCT}(\sigma) \text{ has size } n\}|}$$

and define the (*labelled*) *asymptotic probability* of Ω, $l(\Omega)$, as

$$\lim_{n\to\infty} l_n(\Omega),$$

if it exists. We say that a logic or a class of program schemes has a *zero-one law* if every problem Ω (over a relational signature) definable by a sentence of the logic or accepted by a program scheme from the class is such that the asymptotic probability $l(\Omega)$ exists and is equal to either 0 or 1. For any logical sentence or program scheme ϕ (over a relational signature), we define $l(\phi)$ to be $l(\Omega)$ where Ω is the problem defined by ϕ.

The fact that first-order logic has a zero-one law was proven independently by Fagin [15] and by Glebskij et al. [18]. Central to Fagin's proof are the extension axioms. Let σ be a relational signature and let $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ be a tuple of distinct variables. A maximally consistent set t of σ-atoms and negated σ-atoms (including equalities and inequalities) in the variables of $\mathbf{y}$ is called an *atomic type in* $\mathbf{y}$ *over* σ. We write t to denote both the (finite) set t and the quantifier-free first-order formula consisting of

the conjunction of all atoms and negated atoms in t. Let t' be an atomic type in $(\mathbf{y},\mathbf{z})$ over σ, where $\mathbf{z}$ is a tuple of distinct new variables, such that for every σ-structure $\mathscr{A}$:

$$\mathscr{A} \models \forall\mathbf{y}\forall\mathbf{z}(t'(\mathbf{y},\mathbf{z}) \Rightarrow t(\mathbf{y})).$$

We say that $t'(\mathbf{y},\mathbf{z})$ is an *extension* of $t(\mathbf{y})$ and write $t \subseteq t'$. If $t \subseteq t'$ then the (t,t')-*extension axiom* $\rho_{t,t'}$ is defined as

$$\forall\mathbf{y}(t(\mathbf{y}) \Rightarrow \exists\mathbf{z}t'(\mathbf{y},\mathbf{z})).$$

Fagin proved that the asymptotic probability of every extension axiom exists and is equal to 1.

A problem Ω over some (not necessarily relational) signature σ is *closed under extensions* if whenever a σ-structure $\mathscr{A}$ has a sub-structure in Ω then $\mathscr{A} \in \Omega$ (a σ-structure $\mathscr{A}'$ is a *sub-structure* of $\mathscr{A}$ if the universe of $\mathscr{A}'$ is contained in the universe of $\mathscr{A}$, any relation of $\mathscr{A}'$ is the restriction of the corresponding relation of $\mathscr{A}$ to $|\mathscr{A}'|$ and every constant of $\mathscr{A}'$ is the same as the corresponding constant of $\mathscr{A}$). The following theorem is essentially a generalization of Theorem 7.4 of [12] to extensions of first-order logic using a uniform sequence of Lindström quantifiers corresponding to a problem closed under extensions, where this problem need not just involve graphs but can be over any (not necessarily relational) signature.

The following remark is in order before we proceed. It is easy to see that if we consider problems over signatures containing constant symbols, definable in first-order logic, say, we cannot hope to get a zero-one law for such a problem. However, we can extend first-order logic using uniform sequences of Lindström quantifiers corresponding to problems involving constant symbols and talk about whether the resulting logic has a zero-one law. Of course, the problems we define in such a logic will always be over relational signatures. Path system logic is a case in point. The problem PS is over a signature involving constant symbols but the logic $(\pm\mathrm{PS})^*[\mathrm{FO}]$ is a fragment of bounded-variable infinitary logic; and so by [29] any problem *over a relational signature* definable in path system logic has a zero-one law.

Theorem 15. *Let Ω be a problem closed under extensions. Then the logic $(\pm\Omega)^*[\mathrm{FO}]$ has a zero-one law.*

Proof. Suppose that the problem Ω is over the signature $\sigma_0 = \langle R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c\rangle$, where each R_i is a relation symbol of arity a_i and each C_j is a constant symbol.

Let us begin by considering a formula Ψ of $(\pm\Omega)^*[\mathrm{FO}]$ of the form

$$\Omega[\lambda\mathbf{x}_1\psi_1(\mathbf{x}_1,\mathbf{y}),\mathbf{x}_2\psi_2(\mathbf{x}_2,\mathbf{y}),\ldots,\mathbf{x}_r\psi_r(\mathbf{x}_r,\mathbf{y})](\mathbf{z}_1,\mathbf{z}_2,\ldots,\mathbf{z}_c),$$

whose underlying signature σ is relational and where:

- all free variables of Ψ are contained in $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ and all variables in the k-tuples $\mathbf{z}_1,\mathbf{z}_2,\ldots,\mathbf{z}_c$ are in $\{y_1, y_2, \ldots, y_m\}$; and

- $|\mathbf{x}_i| = ka_i$ and ψ_i is quantifier-free first-order, for every $i = 1, 2, \ldots, r$.

We would like to prove that the asymptotic probability of the sentence

$$\forall \mathbf{y} \left(\bigwedge \{ y_i \neq y_j \colon\ 1 \leqslant i < j \leqslant m \} \Rightarrow (\Psi(\mathbf{y}) \Leftrightarrow \theta(\mathbf{y})) \right)$$

is 1, for some quantifier-free first-order formula $\theta(\mathbf{y})$. Let $t_1, t_2, \ldots, t_p$ be an enumeration of the distinct atomic types in $\mathbf{y}$ over σ containing the negated atoms $\{ y_i \neq y_j \colon 1 \leqslant i < j \leqslant m \}$. Suppose that for every $i = 1, 2, \ldots, p$, either the asymptotic probability of the sentence

$$\forall \mathbf{y}(t_i(\mathbf{y}) \Rightarrow \Psi(\mathbf{y}))$$

is 1 or the asymptotic probability of the sentence

$$\forall \mathbf{y}(t_i(\mathbf{y}) \Rightarrow \neg \Psi(\mathbf{y}))$$

is 1. We shall show that this suffices to achieve our aim.

Let $I \subseteq \{1, 2, \ldots, p\}$ be the set of indices for which the former holds and $J = \{1, 2, \ldots, p\} \backslash \{I\}$, the set of indices for which the latter holds. Then the asymptotic probability of the sentence

$$\bigwedge_{i \in I} \forall \mathbf{y}(t_i(\mathbf{y}) \Rightarrow \Psi(\mathbf{y})) \wedge \bigwedge_{j \in J} \forall \mathbf{y}(t_j(\mathbf{y}) \Rightarrow \neg \Psi(\mathbf{y}))$$

is 1; that is, the asymptotic probability of the sentence

$$\forall \mathbf{y} \left(\bigwedge_{i \in I} (t_i(\mathbf{y}) \Rightarrow \Psi(\mathbf{y})) \wedge \bigwedge_{j \in J} (t_j(\mathbf{y}) \Rightarrow \neg \Psi(\mathbf{y})) \right)$$

is 1; that is, the asymptotic probability of the sentence Γ

$$\forall \mathbf{y} \left(\left(\left(\bigvee_{i \in I} t_i(\mathbf{y}) \right) \Rightarrow \Psi(\mathbf{y}) \right) \wedge \left(\left(\bigvee_{j \in J} t_j(\mathbf{y}) \right) \Rightarrow \neg \Psi(\mathbf{y}) \right) \right)$$

is 1. Consider the sentence Γ'

$$\forall \mathbf{y} \left((t_1(\mathbf{y}) \vee t_2(\mathbf{y}) \vee \cdots \vee t_p(\mathbf{y})) \Rightarrow \left(\Psi(\mathbf{y}) \Leftrightarrow \neg \bigvee_{j \in J} t_j(\mathbf{y}) \right) \right).$$

Given any σ-structure $\mathscr{A}$ and any tuple $\mathbf{a} \in |\mathscr{A}|^m$, the t_i's are mutually exclusive, i.e., at most one of them holds. Hence, Γ and Γ' are logically equivalent; and Γ' is a sentence of the required form whose asymptotic probability is 1.

Fix some atomic type t_i in $\mathbf{y}$ over σ. Suppose that there does not exist any σ-structure $\mathscr{A}$ and $\mathbf{a} \in |\mathscr{A}|^m$ for which $(\mathscr{A}, \mathbf{a}) \models t_i(\mathbf{y})$ and $(\mathscr{A}, \mathbf{a}) \models \Psi(\mathbf{y})$. Then the sentence $\forall \mathbf{y}(t_i(\mathbf{y}) \Rightarrow \neg \Psi(\mathbf{y}))$ is valid.

Hence, suppose that the σ-structure $\mathscr{A}$, of size q, and the tuple $\mathbf{a}=(a_1,a_2,\ldots,a_m)\in|\mathscr{A}|^m$ are such that $(\mathscr{A},\mathbf{a})\models t_i(\mathbf{y})$ and $(\mathscr{A},\mathbf{a})\models\Psi(\mathbf{y})$. Let t' be an atomic type in $(y_1,y_2,\ldots,y_q)$ over σ such that

$$(\mathscr{A},(a_1,a_2,\ldots,a_q))\models t'(y_1,y_2,\ldots,y_q)$$

for some enumeration $a_{m+1},a_{m+2},\ldots,a_q$ of the elements of $\mathscr{A}\backslash\{a_1,a_2,\ldots,a_m\}$. Consequently, $t_i\subseteq t'$. Let the tuple of variables $\mathbf{y}'=(y_{m+1},y_{m+2},\ldots,y_q)$. Note that the asymptotic probability of the (t_i,t')-extension axiom $\forall\mathbf{y}(t_i(\mathbf{y})\Rightarrow\exists\mathbf{y}'t'(\mathbf{y},\mathbf{y}'))$ is 1.

Let the σ-structure $\mathscr{B}$ be such that

$$\mathscr{B}\models\forall\mathbf{y}(t_i(\mathbf{y})\Rightarrow\exists\mathbf{y}'t'(\mathbf{y},\mathbf{y}')).$$

Let $\mathbf{b}\in|\mathscr{B}|^m$ be such that $(\mathscr{B},\mathbf{b})\models t_i(\mathbf{y})$. So, there exists a tuple $\mathbf{b}'\in|\mathscr{B}|^{q-m}$ such that $(\mathscr{B},\mathbf{b},\mathbf{b}')\models t'(\mathbf{y},\mathbf{y}')$. Hence, there is an isomorphic embedding $\iota:(\mathscr{A},\mathbf{a})\hookrightarrow(\mathscr{B},\mathbf{b})$. Consider evaluating $\Psi(\mathbf{y})$ in $(\mathscr{B},\mathbf{b})$. As the formulae $\psi_1,\psi_2,\ldots,\psi_r$ are quantifier-free and Ω is closed under extensions then building the σ-structure described by the formulae $\psi_1,\psi_2,\ldots,\psi_r$ and the tuples $\mathbf{z}_1,\mathbf{z}_2,\ldots,\mathbf{z}_c$ interpreted in the image of $(\mathscr{A},\mathbf{a})$ under the map ι will suffice to tell us that $(\mathscr{B},\mathbf{b})\models\Psi(\mathbf{y})$. Thus, $\mathscr{B}\models\forall\mathbf{y}(t_i(\mathbf{y})\Rightarrow\Psi(\mathbf{y}))$, and the remark at the end of the last paragraph yields that the asymptotic probability of $\forall\mathbf{y}(t_i(\mathbf{y})\Rightarrow\Psi(\mathbf{y}))$ is 1.

A *basic equality type* in $\mathbf{y}$ is an atomic type in $\mathbf{y}$ over the empty signature. Let $\varepsilon(\mathbf{y})$ be a basic equality type different from $\bigwedge\{y_i\neq y_j\colon 1\leqslant i<j\leqslant m\}$. We would also like to prove that the asymptotic probability of the sentence

$$\forall\mathbf{y}(\varepsilon(\mathbf{y})\Rightarrow(\Psi(\mathbf{y})\Leftrightarrow\theta(\mathbf{y})))$$

is 1, for some quantifier-free first-order formula $\theta(\mathbf{y})$. A formula such as that above is equivalent to a formula of the form

$$\forall\mathbf{y}''\left(\bigwedge\{y_i\neq y_j\colon\ 1\leqslant i<j\leqslant m'\}\Rightarrow(\Psi(\mathbf{y}'')\Leftrightarrow\theta(\mathbf{y}''))\right),$$

where $\mathbf{y}''=(y_1,y_2,\ldots,y_{m'})$, for some $m'<m$, which we have essentially already handled. Hence, there exists a quantifier-free first-order formula $\theta(\mathbf{y})$ such that the sentence

$$\forall\mathbf{y}(\Psi(\mathbf{y})\Leftrightarrow\theta(\mathbf{y}))$$

has asymptotic probability 1.

Any sentence of $(\pm\Omega)^*$[FO] is built from atoms or negated atoms by applying first-order operations or the operator Ω so that the total number of operations applied is finite. Hence, by (essentially) replacing applications of the operator Ω with an appropriate quantifier-free first-order formula, we obtain that the asymptotic probability of any sentence of $(\pm\Omega)^*$[FO] is equal to the asymptotic probability of some first-order sentence, which is 0 or 1 by [15, 18]. □

The following corollary is immediate from Corollaries 10 and 15, as the problem *Token Reachability* is closed under extensions.

Corollary 16. *The class of program schemes* NPSA *has a zero–one law.*

6. Conclusions

In this paper we have investigated a naturally defined class of program schemes, NPSA, which take finite structures as inputs, and proven that the class of problems accepted by the program schemes of NPSA coincides with the class of problems defined by the sentences of an extension of first-order logic using a uniform sequence of Lindström quantifiers (corresponding to a **PSPACE**-complete problem). Moreover, we have extended a result due to Dawar and Grädel to enable us to show that the class of program schemes NPSA has a zero-one law and we have applied an existing result due to Stewart to show that there are problems in NPSA which are not definable in bounded-variable infinitary logic. We feel that our general approach of investigating more 'computational versions of logics' (that is, classes of program schemes) than is often the case in finite model theory is completely natural, interesting and novel; and the results presented here and obtained in [2, 6, 43] further testify to this belief.

There remain many unanswered questions concerning classes of program schemes. The most notable ones arising from this paper are: '*Are there problems definable in partial fixed-point logic which are not accepted by any program scheme of* NPSA?'; and '*Is the hierarchy of program schemes* $\mathrm{NPSA}(1) \subseteq \mathrm{NPSA}(2) \subseteq \mathrm{NPSA}(3) \subseteq \cdots$ *proper*?'. We conjecture that the answer to both of these questions is 'Yes'; although so far we have been unable to apply or extend the techniques of [2] (which were used to obtain similar proper hierarchies in the classes of program schemes NPS and NPSS, even on restricted classes of structures) to answer either of these questions.

References

[1] S. Abiteboul, V. Vianu, Generic computation and its complexity, Proc. 23rd Ann. ACM Symp. on Theory of Computing, ACM Press, New York, 1991, pp. 209–219.
[2] A.A. Arratia-Quesada, S.R. Chauhan, I.A. Stewart, Hierarchies in classes of program schemes, J. Logic Computat. 9 (1999) 915–957.
[3] S. Brown, D. Gries, T. Szymanski, Universality of data retrieval languages, Proc. 6th Ann. ACM Symp. on Principles of Programming Languages, ACM Press, New York, 1979, pp. 110–117.
[4] A. Chandra, Programming primitives for database languages, Proc. 8th Ann. ACM Symp. on Principles of Programming Languages, ACM Press, New York, 1981, pp. 50–62.
[5] A. Chandra, D. Harel, Structure and complexity of relational queries, J. Comput. System Sci. 25 (1982) 99–128.
[6] S.R. Chauhan, I.A. Stewart, On the power of built-in relations in certain classes of program schemes, Inform. Process. Lett. 69 (1999) 77–82.
[7] R. Constable, D. Gries, On classes of program schemata, SIAM J. Comput. 1 (1972) 66–118.
[8] S.A. Cook, An observation on time-storage trade off, J. Comput. System Sci. 9 (1974) 308–316.
[9] B. Courcelle, Recursive applicative program schemes, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. B, Elsevier, Amsterdam, 1990, pp. 459–492.
[10] E. Dahlhaus, Reduction to NP-complete problems by interpretations, Lecture Notes in Computer Science, Vol. 171, Springer, Berlin, 1984, pp. 357–365.
[11] A. Dawar, Generalized quantifiers and logical reducibilities, J. Logic Computat. 5 (1995) 213–226.

[12] A. Dawar, E. Grädel, Generalized quantifiers and 0-1 laws, Proc. 10th IEEE Ann. Symp. on Logic in Computer Science, 1995, pp. 54–64.
[13] H.D. Ebbinghaus, J. Flum, Finite Model Theory, Springer, Berlin, 1995.
[14] J. Esparza, M. Nielsen, Decidability issues for Petri nets—a survey, J. Inform. Process. Cybernet. 30 (1994) 143–160.
[15] R. Fagin, Probabilities on finite models, J. Symbolic Logic 41 (1976) 50–58.
[16] H. Friedman, Algorithmic procedures, generalized Turing algorithms and elementary recursion theory, in: R.O. Gandy, C.M.E. Yates (Eds.), Logic Colloquium, 1969, North-Holland, Amsterdam, 1971, pp. 361–390.
[17] M. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[18] Y.V. Glebskij, D.I. Kogan, M.I. Liogon'kij, V.A. Talanov, Range and degree of realizability of formulas in the restricted predicate calculus, Cybernetics 5 (1969) 142–154.
[19] G. Gottlob, Relativized logspace and generalized quantifiers over finite ordered structures, J. Symbolic Logic 62 (1997) 545–574.
[20] E. Grädel, G. McColm, Hierarchies in transitive closure logic, stratified datalog and infinitary logic, Ann. Pure Appl. Logic 77 (1996) 166–199.
[21] M. Grohe, Arity hierarchies, Ann. Pure Appl. Logic 82 (1996) 103–163.
[22] M. Grohe, Existential least fixed-point logic and its relatives, J. Logic Computat. 7 (1997) 205–228.
[23] M.H.T. Hack, Decidability questions for petri nets, Ph.D. Thesis, M.I.T., 1976.
[24] D. Harel, D. Peleg, On static logics, dynamic logics, and complexity classes, Inform. Control 60 (1984) 86–102.
[25] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16 (1987) 760–778.
[26] N.D. Jones, S.S. Muchnik, Even simple programs are hard to analyze, J. Assoc. Comput. Mach. 24 (1977) 338–350.
[27] N.D. Jones, L.H. Landweber, Y.E. Lien, Complexity of some problems in Petri nets, Theoret. Comput. Sci. 4 (1977) 277–299.
[28] Ph.G. Kolaitis, The expressive power of stratified logic programs, Inform. Computat. 90 (1991) 50–66.
[29] Ph.G. Kolaitis, M.Y. Vardi, Infinitary logic and 0-1 laws, Inform. Computat. 98 (1992) 258–294.
[30] D. Kozen, J. Tiuryn, Logics of programs, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. B, Elsevier, Amsterdam, 1990, pp. 789–840.
[31] R.J. Lipton, The reachability problem requires exponential space, Department of Computer Science, Research Report 62, Yale University, 1976.
[32] E.W. Mayr, Persistence of vector replacement systems is decidable, Acta Inform. 15 (1981) 309–318.
[33] F. Neven, M. Otto, J. Tyszkiewicz, J. Van den Bussche, Adding for-loops to first-order logic, Lecture Notes in Computer Science, Vol. 1540, Springer, Berlin, 1999, pp. 58–69.
[34] M. Otto, Bounded Variable Logics and Counting, Lecture Notes in Logic, Vol. 9, Springer, Berlin, 1997.
[35] M. Paterson, N. Hewitt, Comparative schematology, Proc. Project MAC Conf. on Concurrent Systems and Parallel Computation, ACM Press, New York, 1970, pp. 119–128.
[36] I.A. Stewart, Complete problems involving boolean labelled structures and projection translations, J. Logic Computat. 1 (1991) 861–882.
[37] I.A. Stewart, Using the Hamiltonian path operator to capture NP, J. Comput. System Sci. 45 (1992) 127–151.
[38] I.A. Stewart, Logical and schematic characterization of complexity classes, Acta Inform. 30 (1993) 61–87.
[39] I.A. Stewart, Context-sensitive transitive closure operators, Ann. Pure Appl. Logic 66 (1994) 277–301.
[40] I.A. Stewart, Logical description of monotone NP problems, J. Logic Computat. 4 (1994) 337–357.
[41] I.A. Stewart, On locating cubic subgraphs in bounded-degree connected bipartite graphs, Discrete Maths. 163 (1997) 319–324.
[42] I.A. Stewart, Logics with zero–one laws that are not fragments of bounded-variable infinitary logic, Math. Logic Quart. 41 (1997) 158–178.
[43] I.A. Stewart, Using program schemes to logically capture polynomial-time on certain classes of structures, University of Leicester Technical Report 1998/13 (1998).
[44] J. Tiuryn, P. Urzyczyn, Some relationships between logics of programs and complexity theory, Theoret. Comput. Sci. 60 (1988) 83–108.